REMARKS

I. INTRODUCTION

Claims 1-3, 5, 8-10, 13-23 and 25-27 are pending in the present application. In view of the following remarks, it is respectfully submitted that all of the presently pending claims are allowable.

II. THE 35 U.S.C. § 103(a) REJECTIONS SHOULD BE WITHDRAWN

Claims 1-3, 5, 8-10, 13-20, 21-23 and 25-27 stand rejected under 35 U.S.C. § 103(a) as unpatentable over U.S. Patent No. 5,359,721 to Kempf et al. (hereinafter "Kempf") in view of U.S. Patent No. 4,430,705 to Cannavino et al. (hereinafter "Cannavino") in view of U.S. Patent No. 6,256,657 to Chu (hereinafter "Chu") (see 8/29/05 Office Action, pp. 2-8).

Kempf describes a method of allowing a process executing in non-supervisor mode to perform dynamic linking across address spaces without compromising system security (see Kempf, Col. 2, ll. 50-52). Kempf compares this to a typical process executing in non-supervisor mode, which cannot access another process executing in another address space without invoking the operating system (see Kempf, Col. 1, ll. 24-27). Although a process can dynamically link shared libraries into itself when it starts up without entering supervisor mode, security may be compromised (see Kempf, Col. 2, ll. 23-29).

Cannavino describes an authorization mechanism for establishing addressability to information in another address space. This mechanism permits a program in one address space to access data in another address space without invoking a supervisor (see Cannavino, Abstract). A subsystem control facility provides basic authority control with dual address space memory references, program subsystem linkages, and Address Space Number ("ASN") translation to main memory addresses with authorization control (see Cannavino, Col. 3, ll. 9-14). Basic authority control makes a level of privilege or authority available to problem programs (see Cannavino, Col. 3, ll. 15-16). The dual address space concept provides the problem program with the ability to move information from one address space into another, and also to specify in which address space the operands of the program are to be accessed (see Cannavino, Col. 3, ll. 53-57). This incorporates a segment table, which is used to translate virtual addresses of a particular address space (see Cannavino, Col. 3, ll. 57-64). The subsystem linkage control provides for direct linkage between problem programs by authorizing the execution of a Program Call and Program Transfer instruction (see Cannavino, Col. 4, ll. 17-24). The ASN feature provides translation tables and authorization controls whereby a program in the problem state can designate an address space as being a primary address space or a secondary address space (see Cannavino, Col. 4, l. 66 - Col. 5, l. 2).

Chu describes a method for transferring data held within the kernel space to the user space by remapping the virtual memory areas of the kernel and user spaces via the use of an intermediate memory area (see Chu, Col. 6, ll. 21-34). As the Examiner points out, this is a well-known principle for protection domains, and conventional virtual memory address remapping supports such attachment (see 8/29/05 Office Action, p. 4, ¶ 7).

The Examiner rejected claim 1 as obvious in view of Kempf and in further view of Cannavino and Chu (see 8/29/05 Office Action, p. 2). In support of this rejection, the Examiner points to Kempf's disclosure of an address space object which provides an interface for performing limited operations on the object's address space on behalf of client processes executing in non-supervisor mode (see 8/29/05 Office Action, p. 2, ¶ 4). More generally, a program code manager provides an object oriented interface to a client process, thereby facilitating access to a program code segment object comprising executable binary code of a program (see Kempf, Col. 6, ll. 42-46). This process may be categorized as one in which a user application interacts with another user application. However, Kempf is **not** included in the category in which a user application interacts with a kernel space, because such a process according to Kempf would require switching into supervisor mode (see Kempf, Col. 6, ll. 60-64)

. In contrast, the present invention, as recited in amended claim 1, falls into a category where a user application interacts with a kernel space. Specifically, claim 1 recites "loading a code module into a memory space of a first domain, wherein the first domain owns one of a kernel space and a portion of a user space, the code module including an instruction having a symbol reference"; "determining if the external location is within a second domain that is within a protection view of the first domain, wherein the second domain owns the other one of the kernel space and a portion of the user space"; "requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain; and attaching the second domain to the first domain using an attachment mechanism." Accordingly, claim 1 differs significantly from the teachings of Kempf, since the operations of each respective process occur within differing memory locations.

The Examiner also acknowledges that Kempf fails to disclose a method of handling potential irregularities "including exception handling for unauthorized accesses and attachment of separate address spaces" (see 8/29/05 Office Action, p. 3, ¶ 7). However, the Examiner contends that these deficiencies are cured by Cannavino (*Id.*). Specifically, the Examiner points to Cannavino's disclosure of a method by which a problem program executing in a first address space obtains access to data in a second address space by executing a Set Second ASN ("SSAR") instruction (see 8/29/05 Office Action, p. 3, ¶ 5). In this method, each address space has an associated set of address translation tables (see Cannavino, Col. 15, ll. 66-68). These tables are used to locate address space control parameters, as well as a third authority table used to perform authorization tests (see Cannavino, Col. 11, ll. 48-54). Primary and secondary table descriptors, which point to the first and second address spaces respectively, are stored in their respective control registers (see Cannavino, Col. 15, l. 62 - Col. 16, l. 3). As the program executing in the first address space attempts to access data in the second address space, the descriptor pointing to the second address space is stored into the secondary control register (see Cannavino, Col. 15, l. 68 - Col. 16, l. 3). The address in the secondary control register may then be compared to the address in the primary control register to determine if an address translation operation must be performed to obtain data in the other address space (see Cannavino, Col. 16, ll. 3-10).

Claim 1 of the Applicants' invention recites "determining if the external location is within a second domain that is within a protection view of the first domain, wherein the second domain owns the other one of the kernel space and a portion of the user space." The Examiner acknowledges that neither Kempf nor Cannavino reference a protection view (see 8/29/05 Office Action, p. 3, ¶ 7). However, the Examiner equates the protection view of the present invention with the address space tables described in Cannavino (*Id.*). Each protection domain of the present invention includes a mechanism allowing tasks executing in one protection domain to access resources and objects in a separate protection domain (see Application, p.11, ll. 5-7). One such mechanism is a protection view, included in each protection domain, which defines system resources and objects to which it has access (see Application, p.11, ll. 7-9). A second domain may be found in the protection view of a first domain if the first domain has been attached to the second domain (see Application, p.11, ll. 11-14). In contrast, the address space tables, as described in Cannavino, are merely references which indicate the particular virtual address corresponding to an ASN. Each address translation table does not, however, specify the objects that are accessible by the domain by which the table is owned. Accordingly, the address translation table of Cannavino is not equivalent to the protection view claimed in the present invention.

The Examiner contends that the memory remapping method taught by Chu is a "well-known organizational principle for protection domains" (see 8/29/05 Office Action, p. 4, ¶ 7), and that it therefore would have been obvious to one of ordinary skill in the art to consider such a method in conjunction with Kempf and Cannavino. However, Chu teaches that in order to transfer data from the kernel domain to the user domain, the operating system must first remap the data to the virtual memory area of an intermediate domain, before eventually remapping it to the virtual memory area of the user domain (see Chu, Col. 6, l.56 - Col. 7, l. 6). Chu describes this process as "unsatisfactory" (see Chu., Col. 7, l. 14) because it requires significantly more work than simply remapping virtual memory, and because this additional work makes the total remapping process expensive (see Chu, Col. 1, ll. 58-65). Chu thus teaches that improvements to the conventional data remapping process are desirable (see Chu, Col. 2, ll. 9-13). The method of claim 1 presents such an improvement upon the method described by Chu.

Claim 1 recites "loading a code module into a memory space of a first domain, wherein the first domain owns one of a kernel space and a portion of a user space, the code module including an instruction having a symbol reference"; "determining if the external location is within a second domain that is within a protection view of the first domain, wherein the second domain owns the other one of the kernel space and a portion of the user space"; "requesting attachment of the second domain to the first domain when the second domain is determined not to be within the protection view of the first domain; and attaching the second domain to the first domain using an attachment mechanism." In other words, claim 1 presents a data transfer method wherein a first domain, which owns either kernel space or user space, attaches directly to a second domain, which owns the other of kernel space and user space, in order that the first domain might access resources belonging to the second domain. Unlike the method taught by Chu, Applicants' claimed method does not require the use of an intermediate domain in order for the user domain and the kernel domain to interact. This direct relationship has the effect of eliminating a portion of the work cited by Chu as making data remapping an undesirable and expensive process, and doing so in a different manner than that advocated by Chu.

For at least the reasons recited above, Kempf does not teach or suggest all the limitations of claim 1, and Cannavino and Chu fail to cure these deficiencies. Accordingly, Applicants respectfully request that the rejection of claim 1 be withdrawn. Because claims 2-3 and 5 depend from, and therefore include the limitations of, claim 1, it is respectfully submitted that these claims are also allowable.

The Examiner rejected claim 8 as unpatentable over Kempf in view of Cannavino and Chu (see 8/29/05 Office Action, pp. 4-5, ¶¶ 8-10). Claim 8 recites similar limitations to those of amended claim 1. Accordingly, it is respectfully submitted that claim 8 should also be allowed for at least the reasons set forth above with respect to claim 1. Because claims 9-10 and 13-14 depend from, and therefore include the limitations of, claim 8, it is respectfully submitted that these claims are also allowable.

The Examiner rejected claim 15 as unpatentable over Kempf in view of Cannavino and Chu (see 8/29/05 Office Action, p. 6, ¶¶ 11-13). Claim 15 recites "a system space having a number of memory locations, wherein the system space includes a kernel space and at least one user space; and a number of protection domains, at least one of the number of protection domains owning a portion of the system space." To support this rejection, the Examiner points to the same disclosure of Kempf, cited above, that was used as a ground for rejection of claims 1 and 8 (see 8/29/05 Office Action, p. 6, ¶¶ 11). As previously mentioned, the system described in Kempf is one in which a user application interacts with another user application. Accordingly, Kempf does not disclose a system space, including a kernel space, wherein a protection domain owns a portion of the system space. Cannavino fails to cure this deficiency. Claim 15 further recites "wherein each of the number of protection domains includes a protection view defining a set of the number of protection domains to which unprotected access may be made." As mentioned above, Cannavino fails to disclose a protection view, or an equivalent thereof. As further support for this rejection, the Examiner relies upon the same application of Chu as was relied upon for the rejections of independent claims 1 and 8. However, the language used by the Examiner in rejecting claim 15 ("Chu teaches the invention as claimed, wherein the first domain owns one of a kernel space and a portion of a user space and the second domain owns the other one of the kernel space and a portion of the user space", see 8/29/05 Office Action, p. 6, ¶ 13) is inapplicable to claim 15 ("a system space having a number of memory locations, wherein the system space includes a kernel space and at least one user space"). Therefore, it is respectfully submitted that claim 15 should be allowed. Because claims 16-23 and 25-27 depend from, and therefore include the limitations of, claim 15, it is respectfully submitted that these claims are also allowable.

CONCLUSION

In light of the foregoing, Applicants respectfully submit that all of the now pending claims are in condition for allowance. All issues raised by the Examiner having been addressed, an early and favorable action on the merits is earnestly solicited.

Respectfully submitted,

Dated: November 29, 2005

By: [illegible]

Michael J. Marcin (Reg. No. 48,198)

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, New York 10038
Tel: (212) 619-6000
Fax: (212) 619-0276